Automatic Python Programming using Stack-based Genetic Programming

Hyun Soo Park
Dept. of Computer Engineering
Sejong University
Seoul, Republic of Korea
+82-2-3408-3838
hspark@sju.ac.kr

Kyung Joong Kim *
Dept. of Computer Engineering
Sejong University
Seoul, Republic of Korea
+82-2-3408-3838
kimkj@sejong.ac.kr

ABSTRACT
Traditional genetic programming uses tree-like data structure to represent a program. It should be converted into a Lisp code, or needs a custom-made virtual machine or interpreter to execute the program generated. Recently, there is a study on genetic programming directly using Java bytecode, a practical intermediate language. It evolves a series of commands that manipulate stack and registers in the virtual machine and represents them with a simple list data structure instead of tree. Evolving the intermediate language is promising because 1) it is easy to combine an existing program with an automatically generated program, 2) there are several available development tools and environments for the language including virtual machine, decompiler, optimizer and so on, and 3) incorporating the list data structure into the evolutionary algorithm is simple and straightforward. In this research, we propose to evolve bytecode of Python programming language by stack-based genetic programming. Python is a flexible and popular programming language powered by plenty of research tools. For the evolution, we developed representation and genetic operations for the Python language. We report that the proposed method produced successful Python codes for two regression problems.

Categories and Subject Descriptors
I.2.2. [**Artificial Intelligence**]: Automatic Programming – *program synthesis*

General Terms: Algorithms, Design

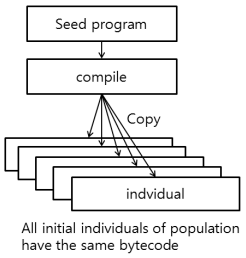
Keywords
Genetic programming, python, bytecode, stack-based genetic programming

1. INTRODUCTION
Automatic programming is one of the ultimate goals of computer science and has been tackled by evolutionary computation. It attempts to build an automated algorithm to generate computer programs which solve the problems. The initial work on evolving computer program is based on the RISC machine code with the genetic programming [1]. Since 1998, there have been several works on evolving JAVA bytecode. Harvey *et al.* solved a functional regression problem [2] and recently, Orlov *et al.* proposed FINCH (Fertile Darwinian Bytecode Harvester), a methodology for evolving Java bytecode, enabling the evolution of unrestricted Java programs [3].

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.
ACM 978-1-4503-1178-6/12/07.

Although the evolution of bytecode is promising, most of works only use JAVA programming language. JAVA is one of the popular languages in the world. However there are several alternatives with easy-to-use properties. For example, Python is one of the alternatives and despite of its simple grammar to learn, it is so powerful to program scientific applications. In this work, we propose to evolve the bytecode of Python programming language to solve regression problems.

2. GENETIC PRGRAMMING USING PYTHON BYTECODE
In the evolution of the Python code, we start from embryonic codes (a seed function) containing useful building blocks for evolution. Because the evolution of codes often produces invalid codes, it is important to check the validity of individuals. If the code is invalid (not executable), it should be rejected from the population. In this paper, we use only crossover operations because mutation is likely to produce invalid codes. The goodness of the code is evaluated on some sampled points for regression problems. We use a compiled seed function to initialize the population (Figure 1). The seed function needs to have useful operators and operands enabling all individuals of the initial population to be executable.




```
def func(x):
   x=cos(sin(log((((x+x)x)*x)/x))
)
   return x
```

Figure 1. A seed function written in Python code

It is still challenging to evolve a Python program in a high-level language form. Instead, each Python program is represented as a list of bytecode. Python allows to extracting the bytecode from the high-level human-readable codes. In this work, we attempt to evolve a program to represent mathematical equations. Fitness is defined as distance between the target function and results of the current individual's bytecode. It randomly selects 20 sampling points from [0,1) and they are inputted to the both of the target and current individual's bytecode. The 'error' is defined as the sum of distances of outputs for the points. The purpose of the evolution is to minimize the error.

* Corresponding Author

We use a simple two-point crossover. Two individuals (A and B) are chosen randomly as parents (Figure 2). A portion of each individual is randomly chosen. A part of individual A is copied into the erased part of individual B. Figure 2 shows an example of the crossover. The length of the selected portion for each individual is randomly chosen from $N(0, 3)$. If total parents number is N, then current population is $2N$ (offspring + parents). In the selection stage, we choose half of them for the next generation.

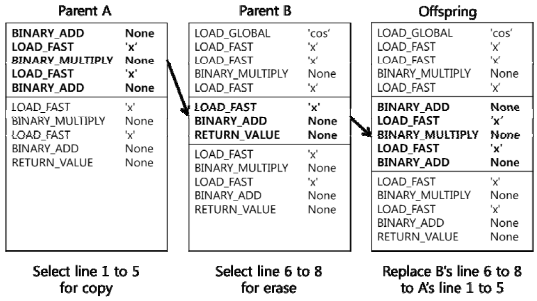


Figure 2. An example of the crossover

3. EXPERIMENTAL RESULTS

In this paper, we applied the proposed Python bytecode evolution to two regression problems (Table 1). We decide to use PyPy (ver 1.6) that implements Python version 2.7 and Byteplay throws exception when it handles with an incorrect bytecode [4][5]. Population size is 1000, maximum number of generation is 50 (problem 1) and 300 (problem 2). Crossover rate is 1.0 and the number of runs is 100. Yield means the percentage of successful runs over the 100 trials.

The yield shows that the easy problem can be solved in most runs and increased from 62% (30 generation) to 94% (94 generations). For the complex function, 30% of runs successfully find the target function (Figure 3). Figure 4 shows an evolved bytecode of problem 1.

Table 1. Target functions and yield of the evolution

	Problem 1	Problem 2
Equations	$y = x^4 + x^3 + x^2 + x$	$y = \sum_{i=1}^{9} x^i$
Yield	62 % at 30 Gen. 94% at 94 Gen.	23 % at 150 Gen. 31 % at 300 Gen.

4. CONCLUSION AND FUTURE WORK

We use the stack-based Genetic Programming to evolve Python bytecode. To the best of our knowledge, it is the first time to evolve the Python in the bytecode level. In this paper, we propose representation and genetic operation for the language. The experimental results on two regression problems show the possibility of the Python-based evolution.

In this work, it is important to define an arbitrary seeding function with some useful building blocks. Because any bytecode missing in the seeding function cannot be used in the evolution, the choice of the function (with enough building blocks) is critical to the success of the evolution. It is desirable to insert new genetic information into the evolution allowing flexible definition of the seeding function.

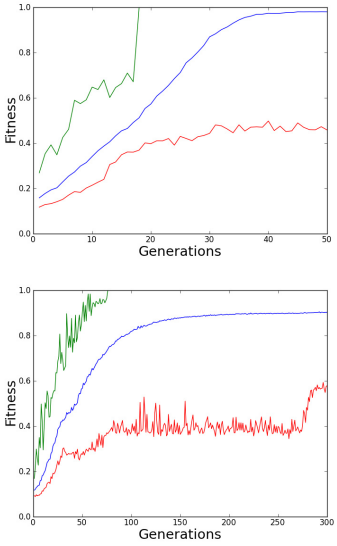


Figure 3. The Changes of fitness (maximum, average, and minimum) (problem 1: top, problem 2: bottom)

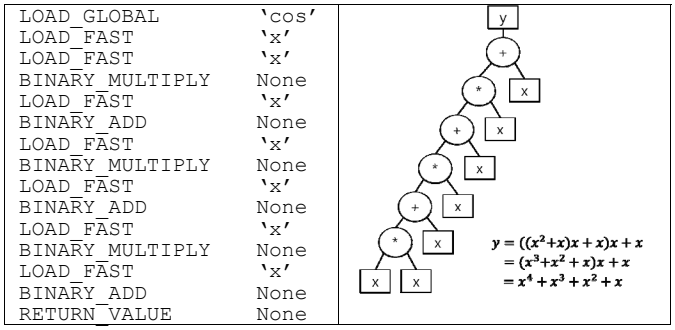


Figure 4. The analysis of a successful Python bytecode evolved for the problem 1

5. ACKNOWLEDGEMENTS

This research was supported by Basic Science Research Program and the Original Technology Research Program for Brain Science through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0012876) (2010-0018948).

6. REFERENCES

[1] Nordin, P., 2004. A compiling genetic programming system that directly manipulates the machine code, *Advances in Genetic Programming*, MIT Press.

[2] Harvey, B., Foster, J. A. and Frincke, D., 1998. Towards byte code genetic programming, *Late Breaking Papers of Genetic Programming*, 1998.

[3] Orlov, M. and Sipper, M. Flight of the FINCH through the Java wilderness, *IEEE Transactions on Evolutionary Computation*, 15, 2, (April. 2011), 166-182.

[4] http://pypy.org/

[5] http://pypi.python.org/pypi/byteplay/0.2